RoleCast
Finding Missing Security Checks When You Do Not Know What Checks Are

Sooel Son, Kathryn S. McKinley,
Vitaly Shmatikov

Mateusz Galimski
May 21, 2012

Table of contents

Introduction

- Web applications interact with untrusted users and receive untrusted network inputs
- security checks prior to executing security-sensitive events
- objective is to develop a robust method for finding missing security checks in web applications

Introduction

- easier if the programmer formally specifies the application's security policy, e.g., via annotations or data-flow assertions

- the overwhelming majority of Web applications today are not accompanied by specifications of their intended authorization policies

Introduction, previous techniques

- syntactic definition of checks as inputs
- must know a priori the syntactic form of every check
- it does not work for finding missing authorization checks in applications because there is no standard set of checks used by all applications
- must infer the set of role-specific checks from the application’s code

Introduction, RoleCast

- automatically infers:
  - the set of user roles
  - the security checks specific to each role
- finds missing security checks, does not rely on programmer annotations or an external specification of intended authorization policy
- does not assume a priori which methods or variables implement security checks

Introduction, RoleCast

- exploits the idea that there is a small number of sources for authorization information (e.g., session state, cookies, results of reading the user database)
- all authorization checks involve a conditional branch on variables holding authorization information
- each page is typically implemented by one or more program files

Introduction, RoleCast

- this approach infers the Web application’s authorization logic under the assumption that the application follows common code design patterns, it may suffer from both false positives and false negatives
- nevertheless, it works well

Introduction, other approaches (1)

- taint checks, taint analysis
    - cross-site scripting
    - SQL injections
    - if (user == ADMIN) {DB query(‘‘DROP TABLE AllUsers’’)}
    - data-flow not control-flow
- explicit security policy
    - not useful enough

Introduction, other approaches (2)

- dynamic analysis
  - there is no guarantee that the set of checks observed during test executions is comprehensive, dynamic analysis may miss checks

- dynamic and static analyses are complementary

Security Logic in Web Applications

- focus on server-side Web applications, which are typically implemented in PHP and JSP
- client-side applications, which are typically implemented in JavaScript are outside the scope

Security Logic in Web Applications

- PHP programs use a flat file structure with a designated main entry point
- a network user can directly invoke any PHP file by providing its name as part of the URL
- if the file contains executable code outside of function definitions, this code will be executed

Security Logic in Web Applications

- JSP (Java Server Pages) is a Java technology for dynamically generating HTML pages
- mixes Java statements with XML and HTML tags
- build on Java, more object-oriented features than PHP
- executes on Java Virtual Machine

Security Logic in Web Applications

- the languages are quite different
- to demonstrate that our approach we provide a generic method for analyzing security of Web applications regardless of the implementation language, we apply our analysis to both JSP and PHP applications

Security Logic in Web Applications

- translating scripting languages into Java is becoming a popular approach because it helps improve performance by taking advantage of mature JVM compilers

- exploit this practice by:
  - converting Web applications into Java class files
  - extending the Soot static analysis framework for Java programs with new algorithms for static security analysis of Web applications

Security Logic in Web Applications

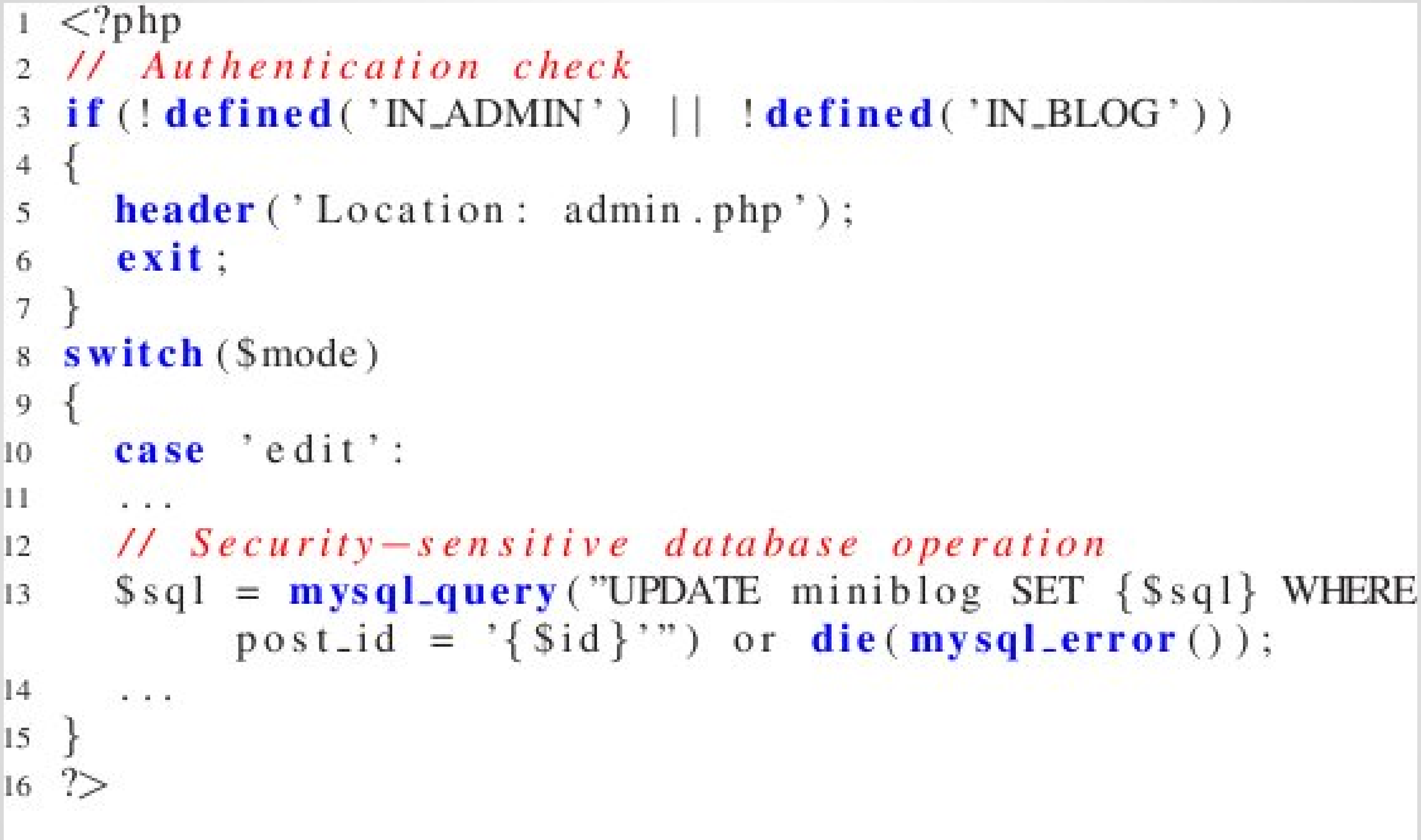
- JSP is translated to Java class files by Tomcat Web Server
  - produces well-formed Java

- PHP is translated by Quercus compiler
  - PHP is a dynamically typed language
  - process of translation obscures the call graph
  - security analysis requires a precise call graph, we must reverse-engineer this translation

Security Logic in Web Applications

- security-sensitive events:
  - all operations that may affect the integrity of database queries that insert, delete, or update the database
  - statically determining the type of a SQL query in a given statement requires program analysis. RoleCast conservatively marks all statically unresolved SQL queries as sensitive
  - SELECT and SHOW queries are deliberately not included

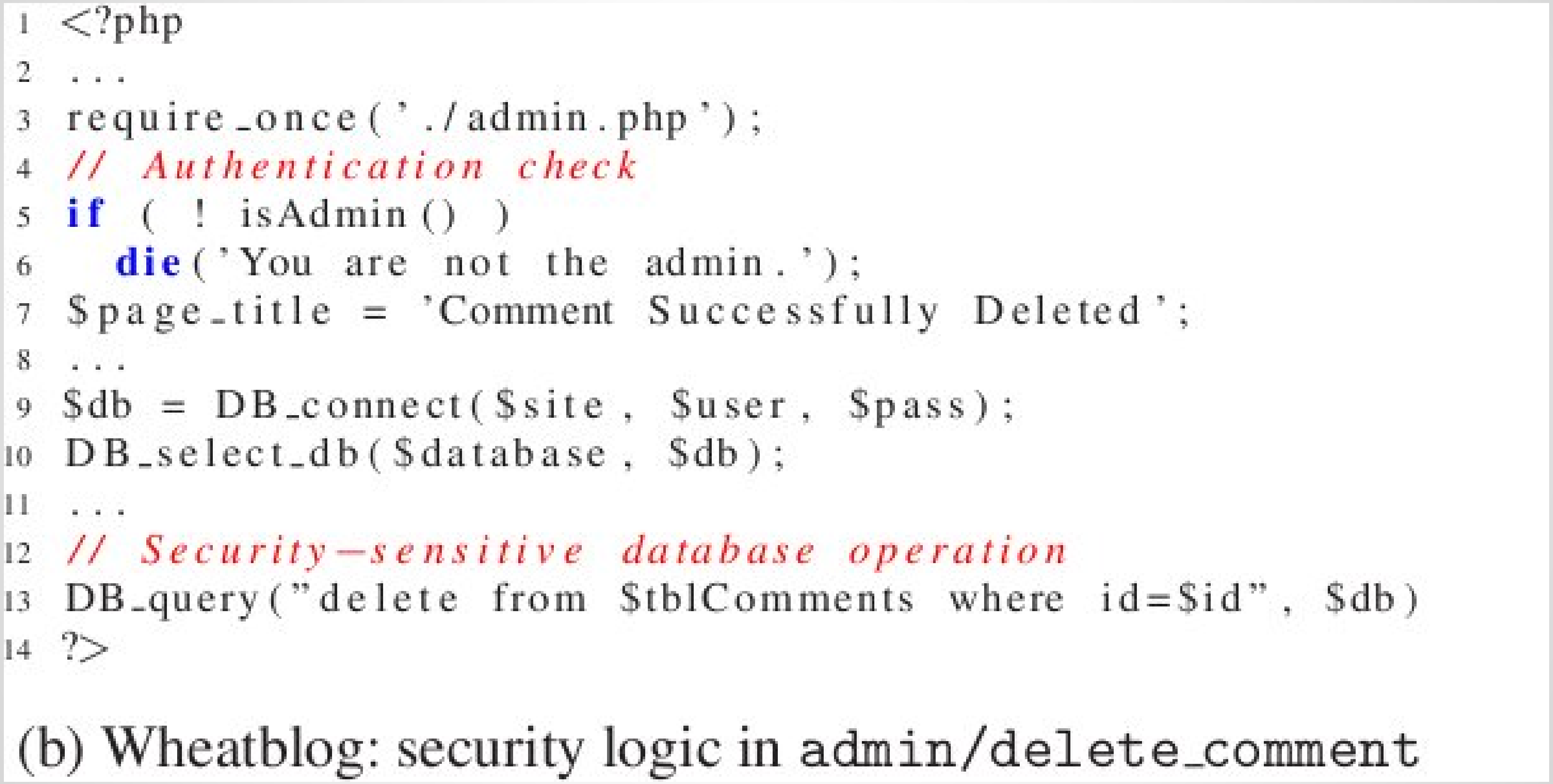
Security Logic in Web Applications, examples (1)



```php
<?php
// Authentication check
if(!defined('IN_ADMIN') || !defined('IN_BLOG'))
{
  header('Location: admin.php');
  exit;
}
switch($mode)
{
  case 'edit':
  ...
  // Security-sensitive database operation
  $sql = mysql_query("UPDATE miniblog SET {$sql} WHERE
       post_id = '{$id}'") or die(mysql_error());
  ...
}
?>
```

(a) Miniblog: security logic in `adm/index.php`

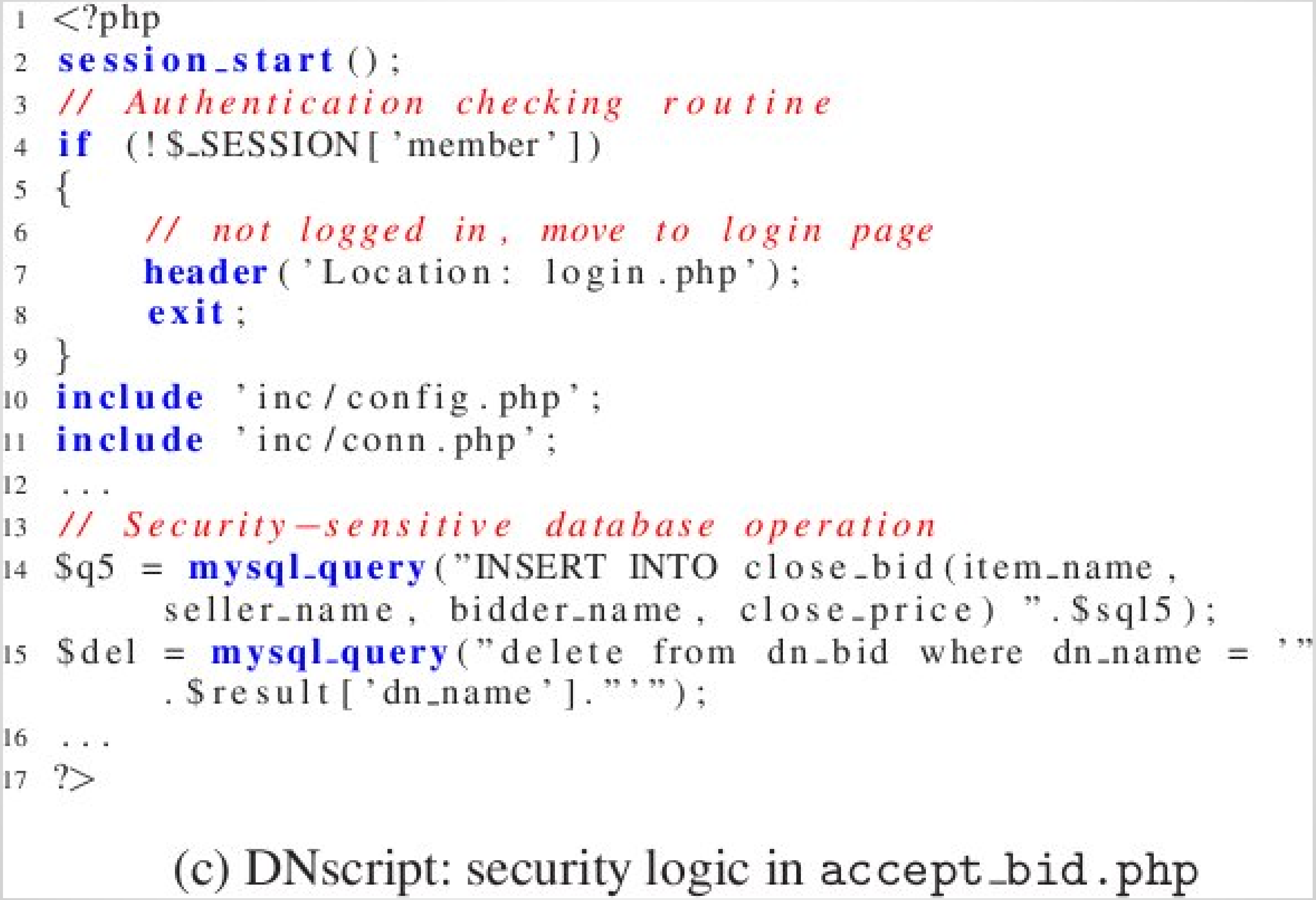
Security Logic in Web Applications, examples (2)



```
<?php
...
require_once('./admin.php');
// Authentication check
if ( ! isAdmin() )
  die('You are not the admin.');
$page_title = 'Comment Successfully Deleted';
...
$db = DB_connect($site, $user, $pass);
DB_select_db($database, $db);
...
// Security-sensitive database operation
DB_query("delete from $tblComments where id=$id", $db)
?>
```

(b) Wheatblog: security logic in admin/delete_comment

Security Logic in Web Applications, examples (3)



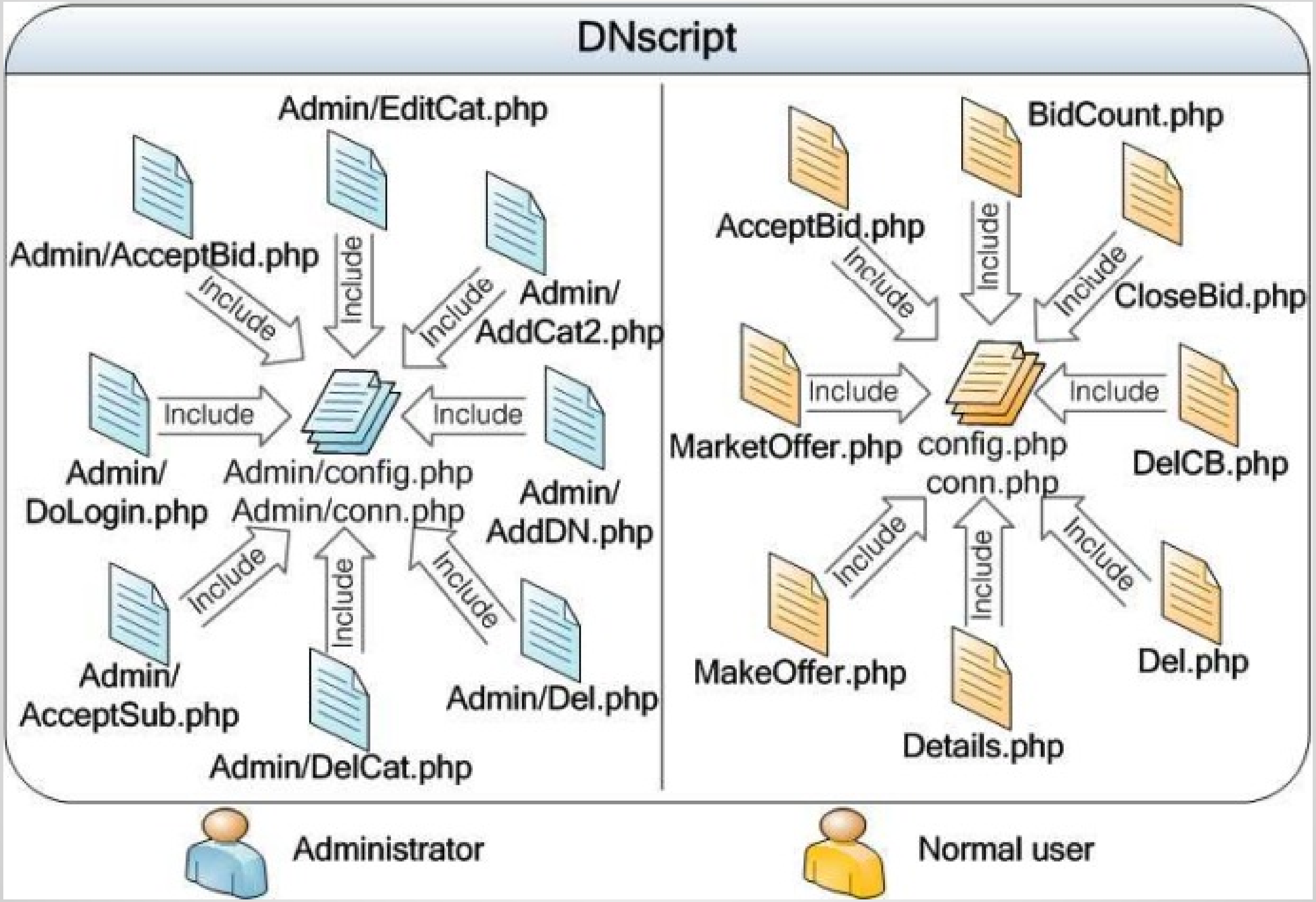
```
<?php
session_start();
// Authentication checking routine
if (!$_SESSION['member'])
{
     // not logged in, move to login page
     header('Location: login.php');
     exit;
}
include 'inc/config.php';
include 'inc/conn.php';
...
// Security-sensitive database operation
$q5 = mysql_query("INSERT INTO close_bid(item_name,
     seller_name, bidder_name, close_price) ".$sql5);
$del = mysql_query("delete from dn_bid where dn_name = '"
     .$result['dn_name']."'");
...
?>
```

(c) DNscript: security logic in accept_bid.php

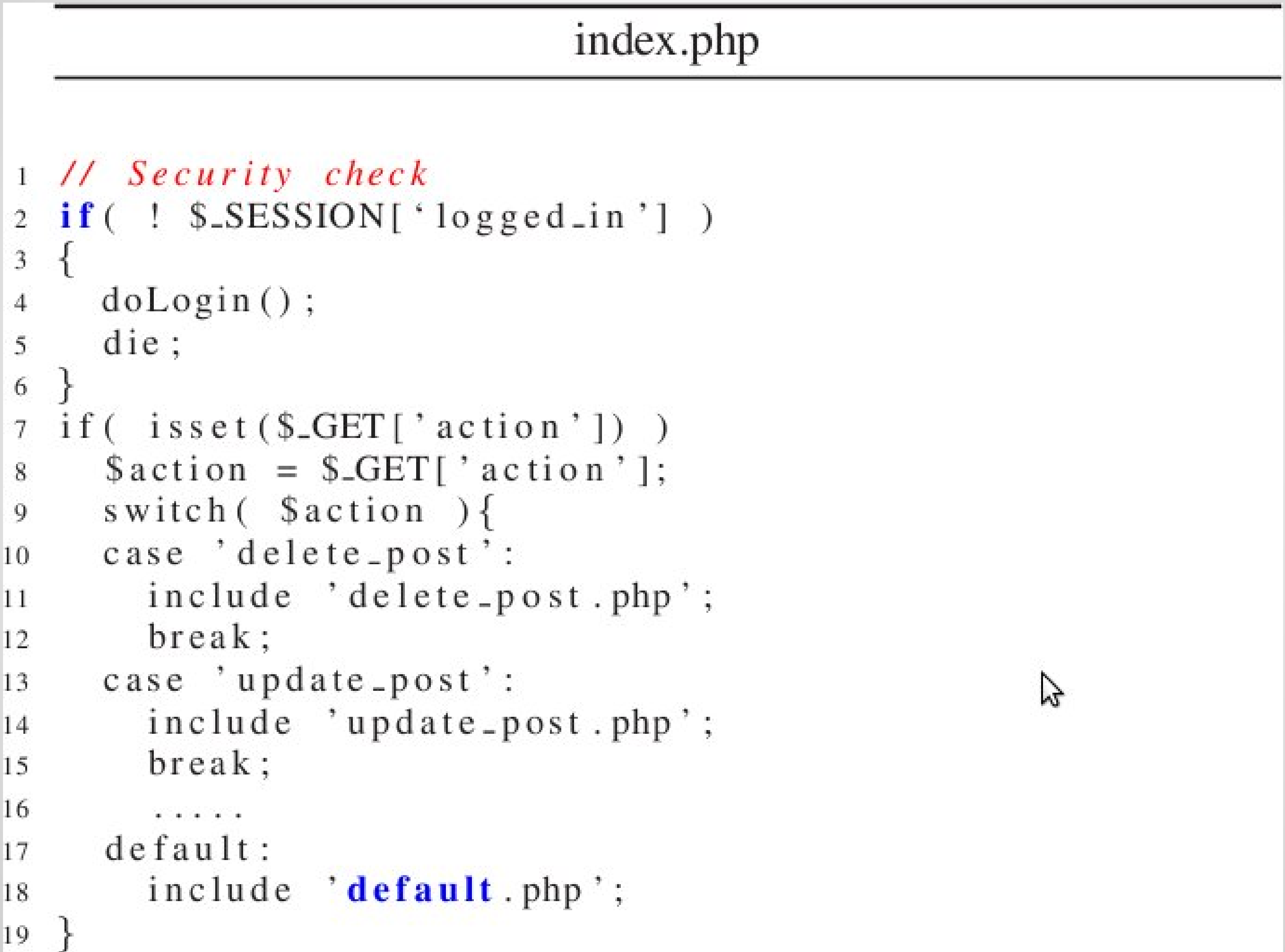
Security Logic in Web Applications, observations

- Important observations:
    - when a security check fails, the program quickly terminates or restarts
    - every path leading to a security-sensitive event from any program entry point must contain a security check
    - distinct application-specific roles usually involve different program files

Security Logic in Web Applications, file structure



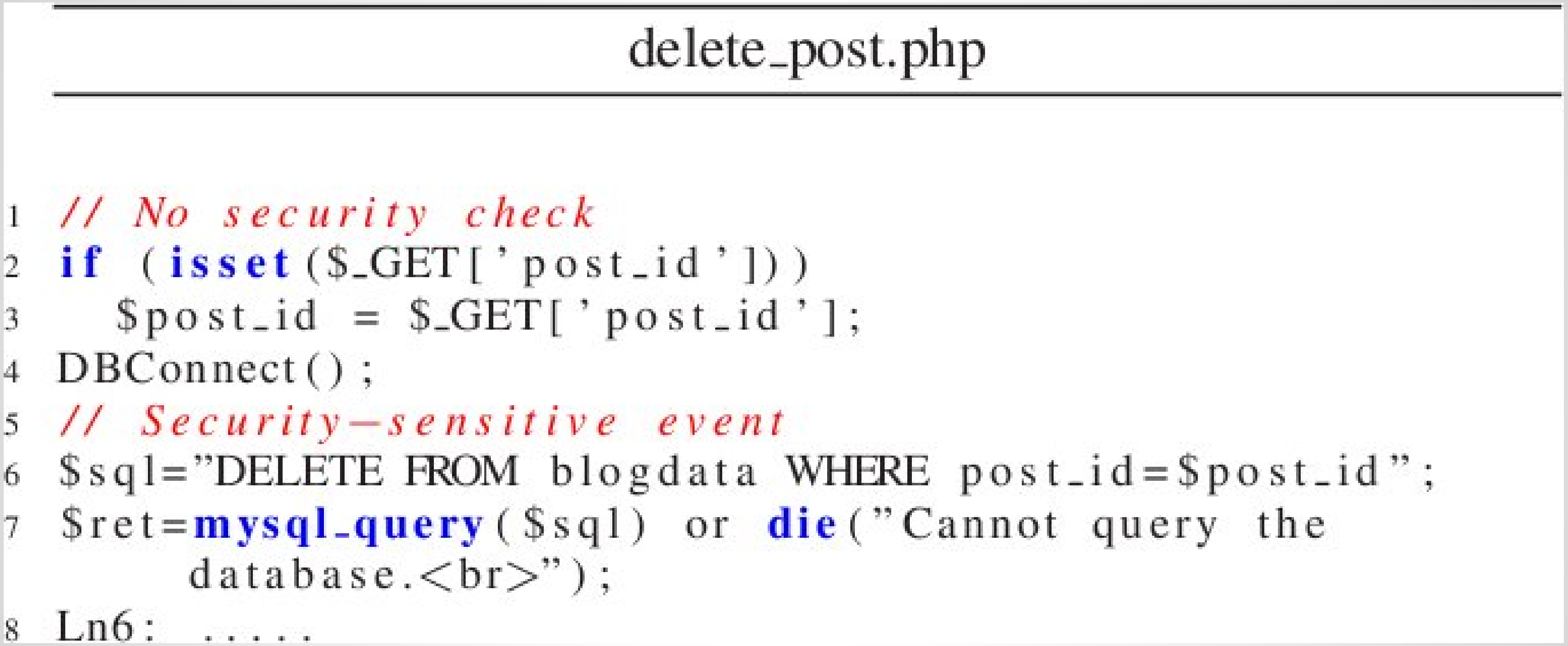
example vulnerability (1)



index.php

```
// Security check
if( ! $_SESSION[‘logged_in’] )
{
  doLogin();
  die;
}
if( isset($_GET['action']) )
  $action = $_GET['action'];
  switch( $action ){
  case 'delete_post':
    include 'delete_post.php';
    break;
  case 'update_post':
    include 'update_post.php';
    break;
    .....
  default:
    include 'default.php';
}
```

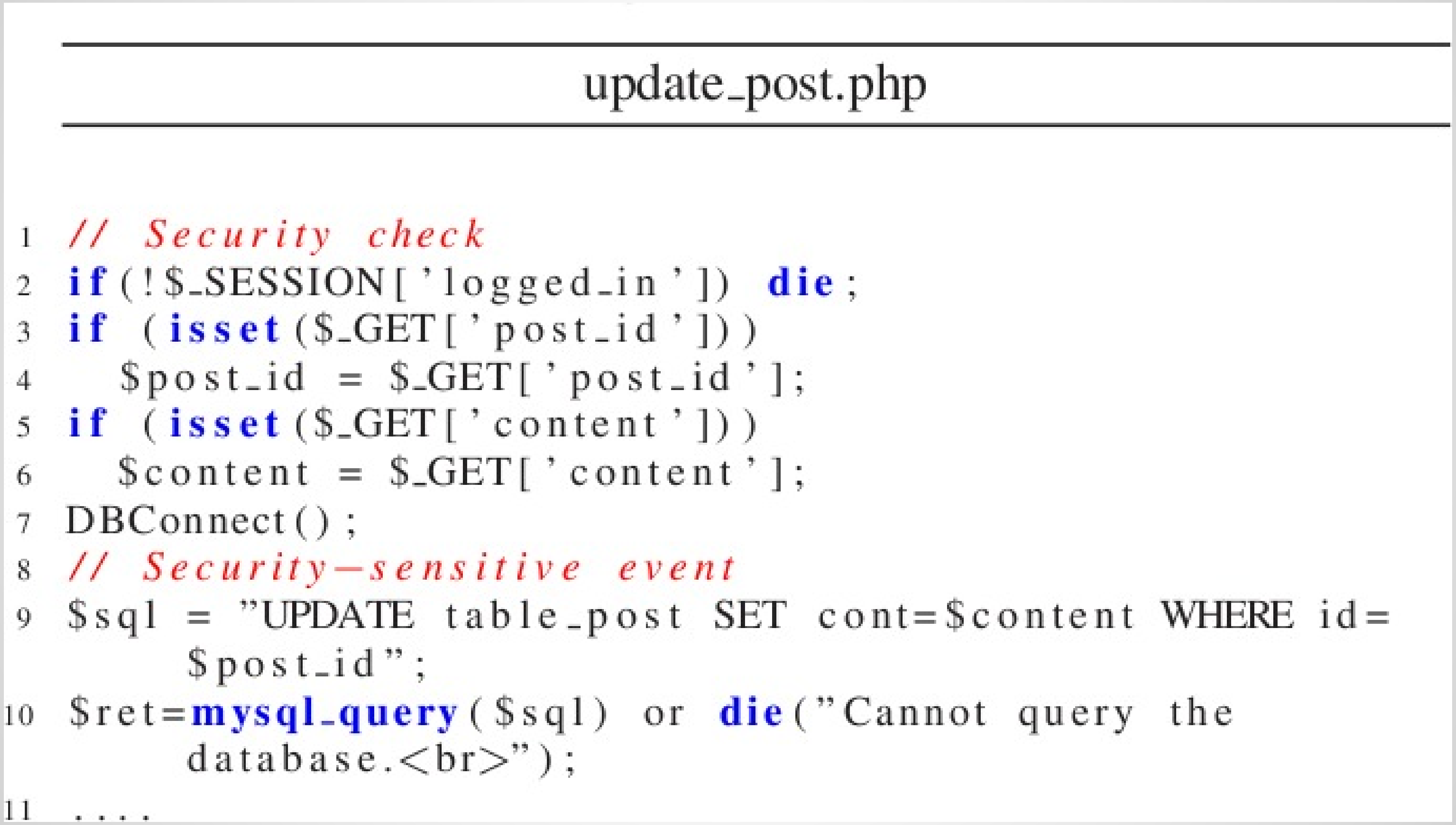
example vulnerability (2)



delete_post.php

```
// No security check
if (isset($_GET['post_id']))
  $post_id = $_GET['post_id'];
DBConnect();
// Security-sensitive event
$sql="DELETE FROM blogdata WHERE post_id=$post_id";
$ret=mysql_query($sql) or die("Cannot query the
     database.<br>");
Ln6: .....
```

example vulnerability (3)



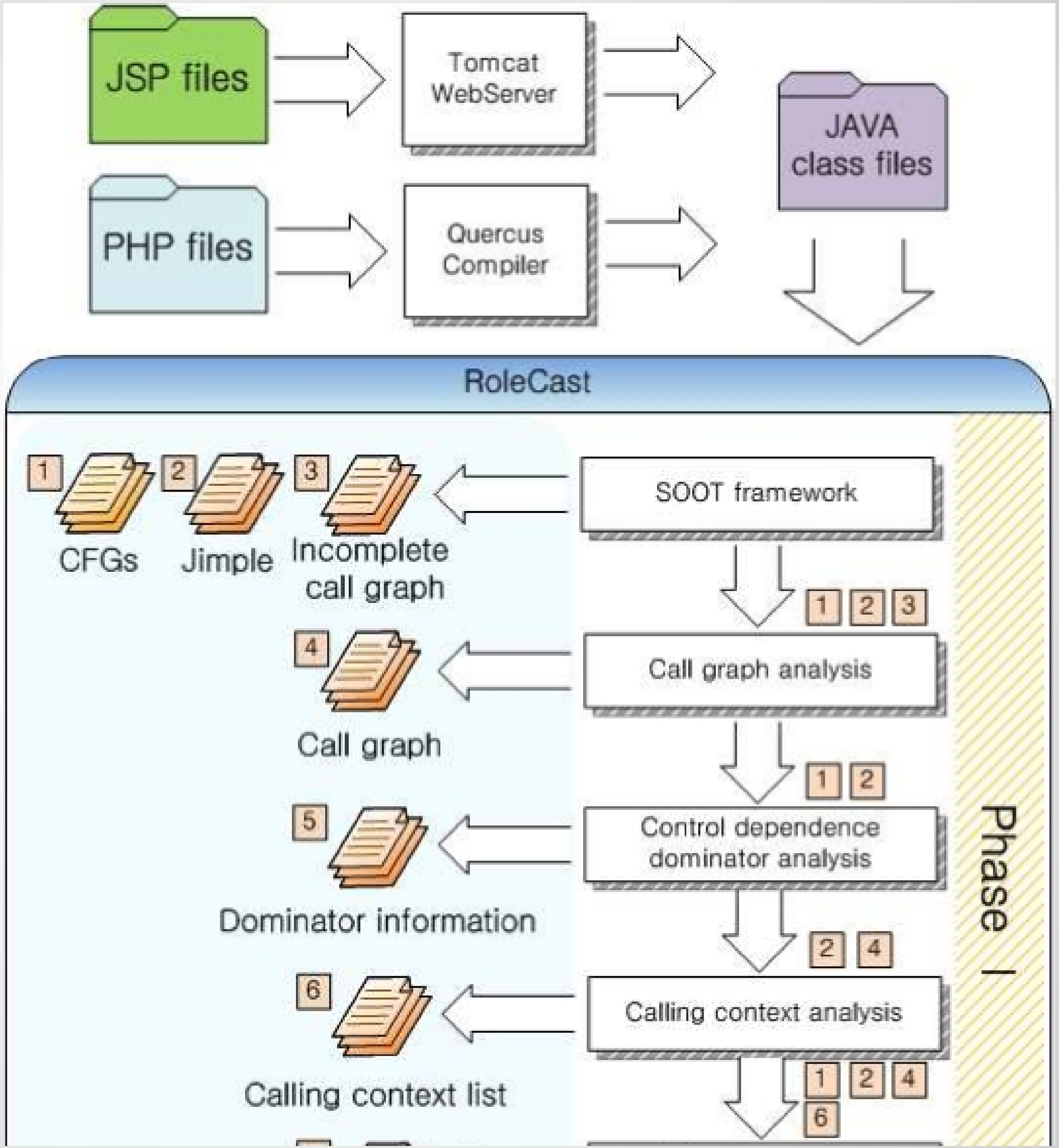
update_post.php

```
// Security check
if(!$_SESSION['logged_in']) die;
if (isset($_GET['post_id']))
  $post_id = $_GET['post_id'];
if (isset($_GET['content']))
  $content = $_GET['content'];
DBConnect();
// Security-sensitive event
$sql = "UPDATE table_post SET cont=$content WHERE id=
      $post_id";
$ret=mysql_query($sql) or die("Cannot query the
      database.<br>");
....
```

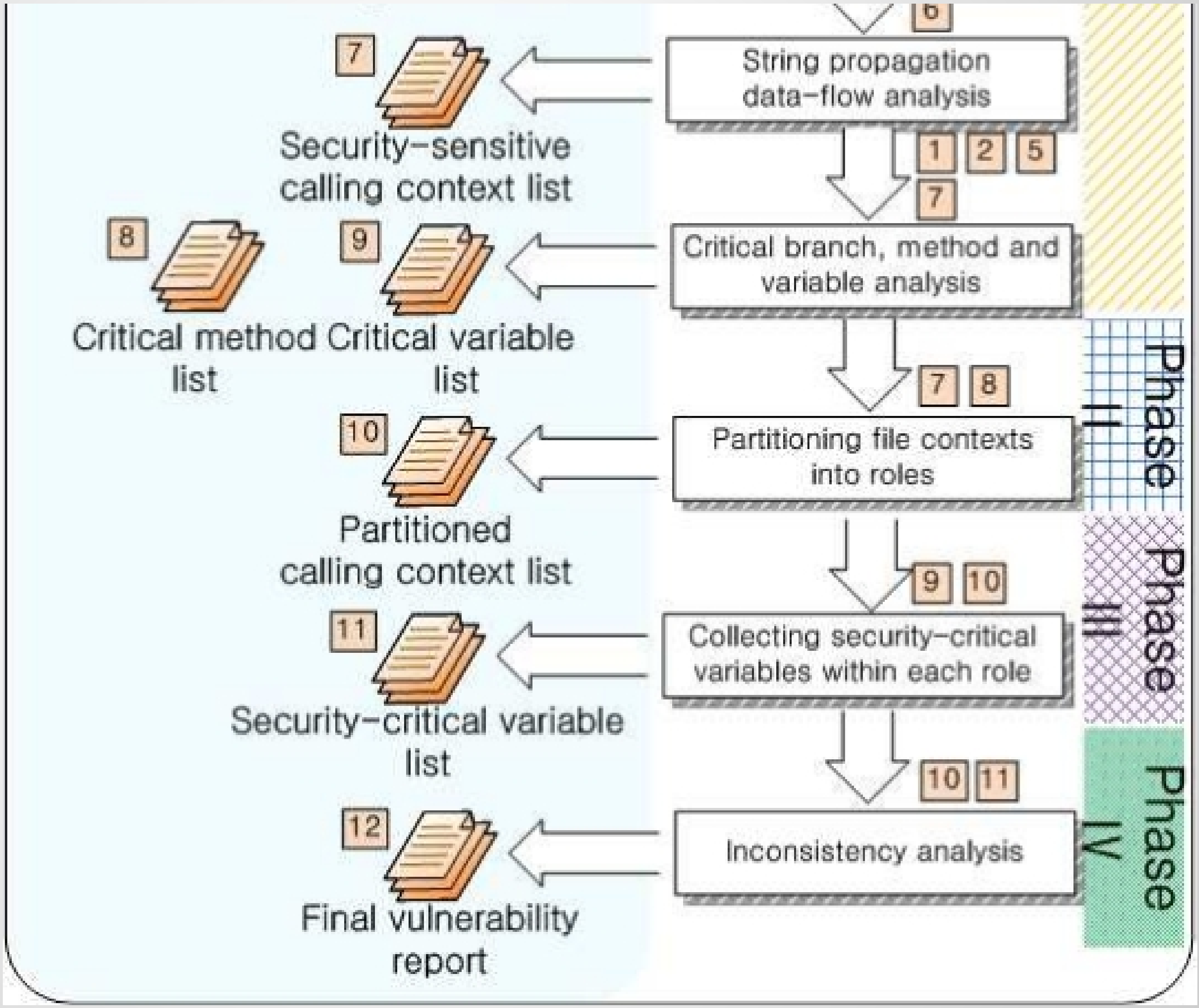
Analysis overview

- RoleCast has four analysis phases:
  - Phase I identifies critical variables that control whether security-sensitive events execute or not
  - Phase II partitions contexts into groups that approximate application-specific user roles
  - Phase III computes for each role the subset of critical variables responsible for enforcing the security logic of that role
  - Phase IV discovers missing security checks by verifying whether the relevant variables are checked consistently within the role

Architecture



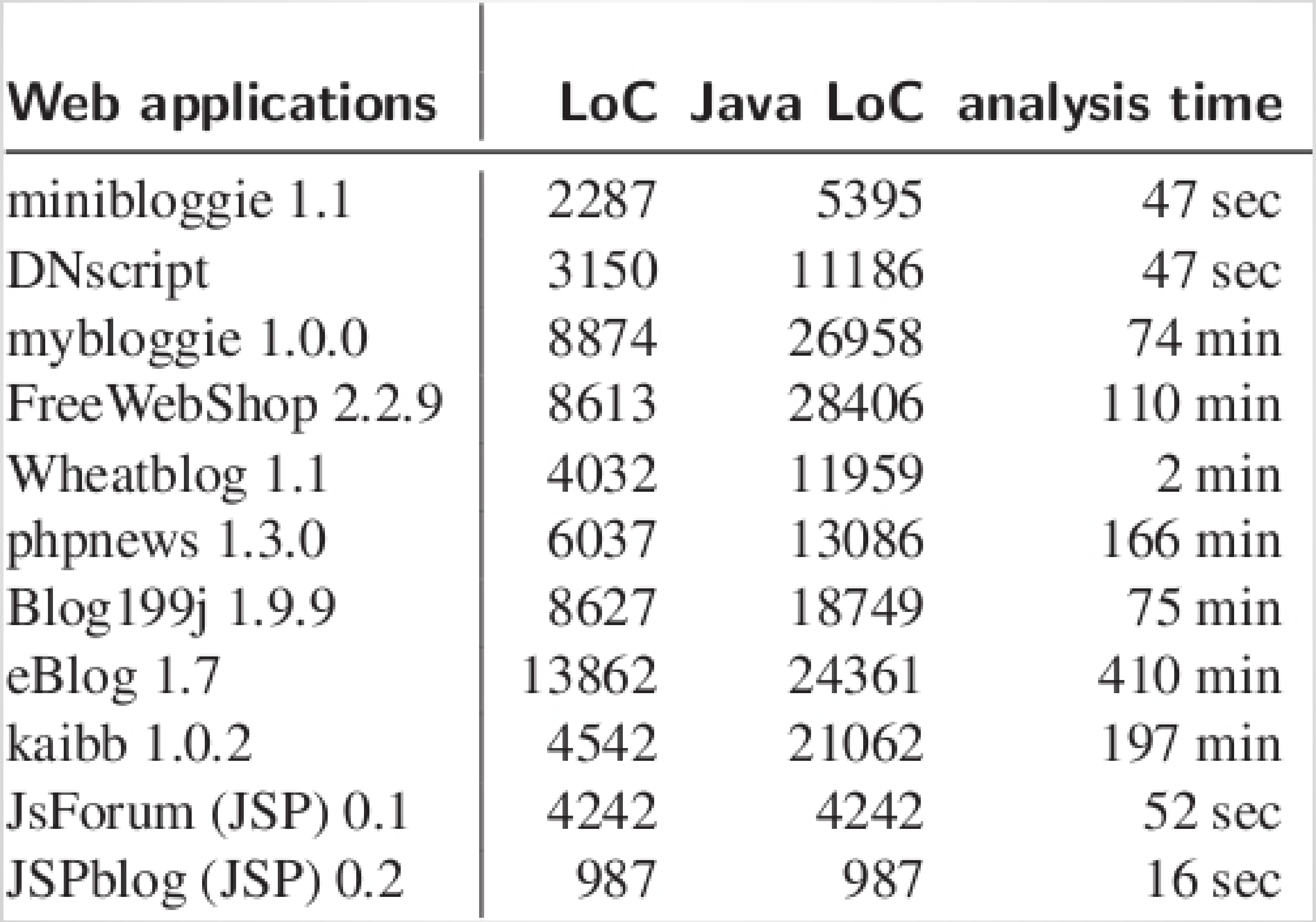
Architecture


6
String propagation data-flow analysis
7
Security-sensitive calling context list
1 2 5
7
Critical branch, method and variable analysis
8
Critical method list
9
Critical variable list
7 8
Partitioning file contexts into roles
10
Partitioned calling context list
9 10
Collecting security-critical variables within each role
11
Security-critical variable list
10 11
Inconsistency analysis
12
Final vulnerability report
Phase II
Phase III
Phase IV

Experimental Evaluation

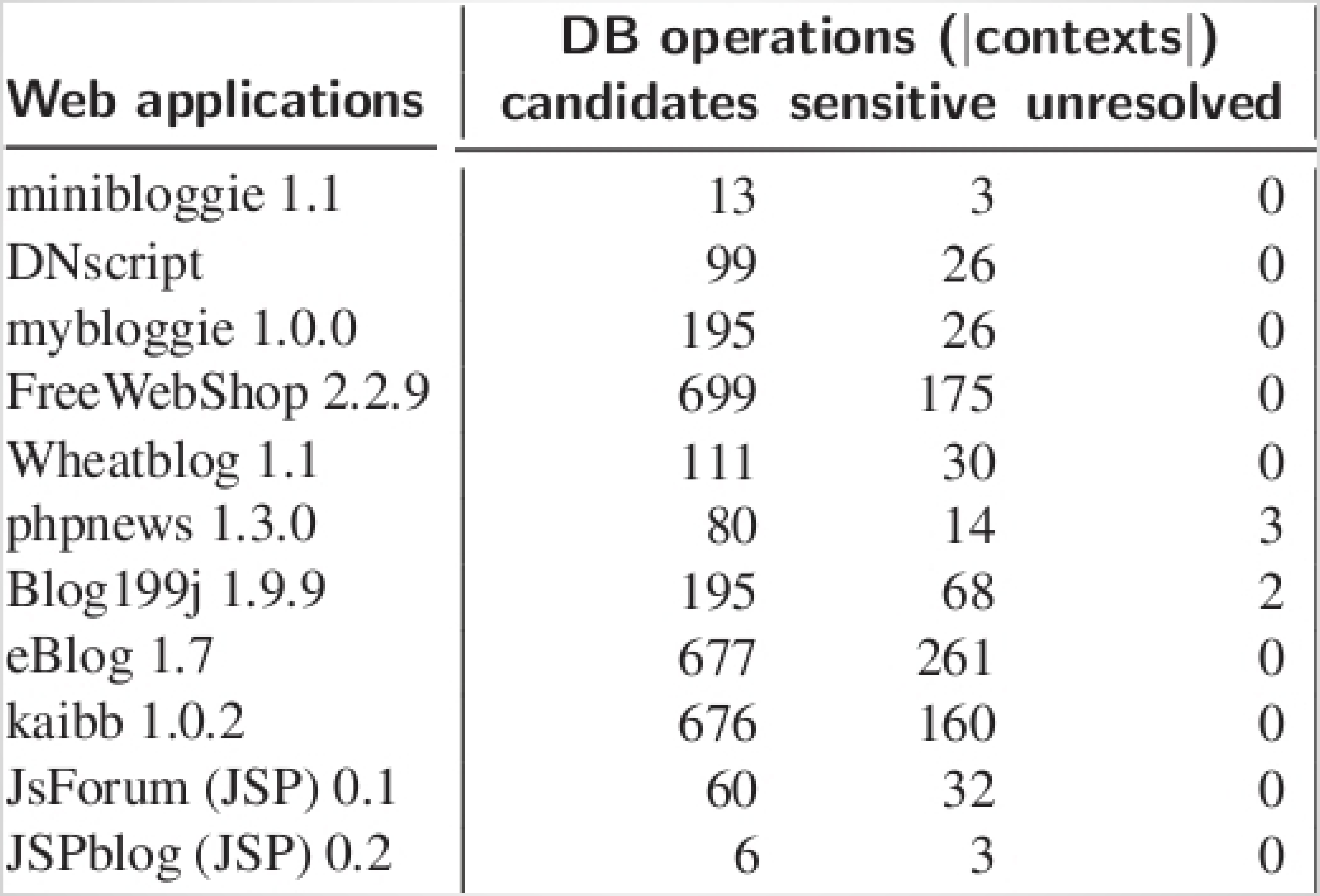
- all experiments in this section were performed on a Pentium 3GHz with 2G of RAM

Experimental Evaluation



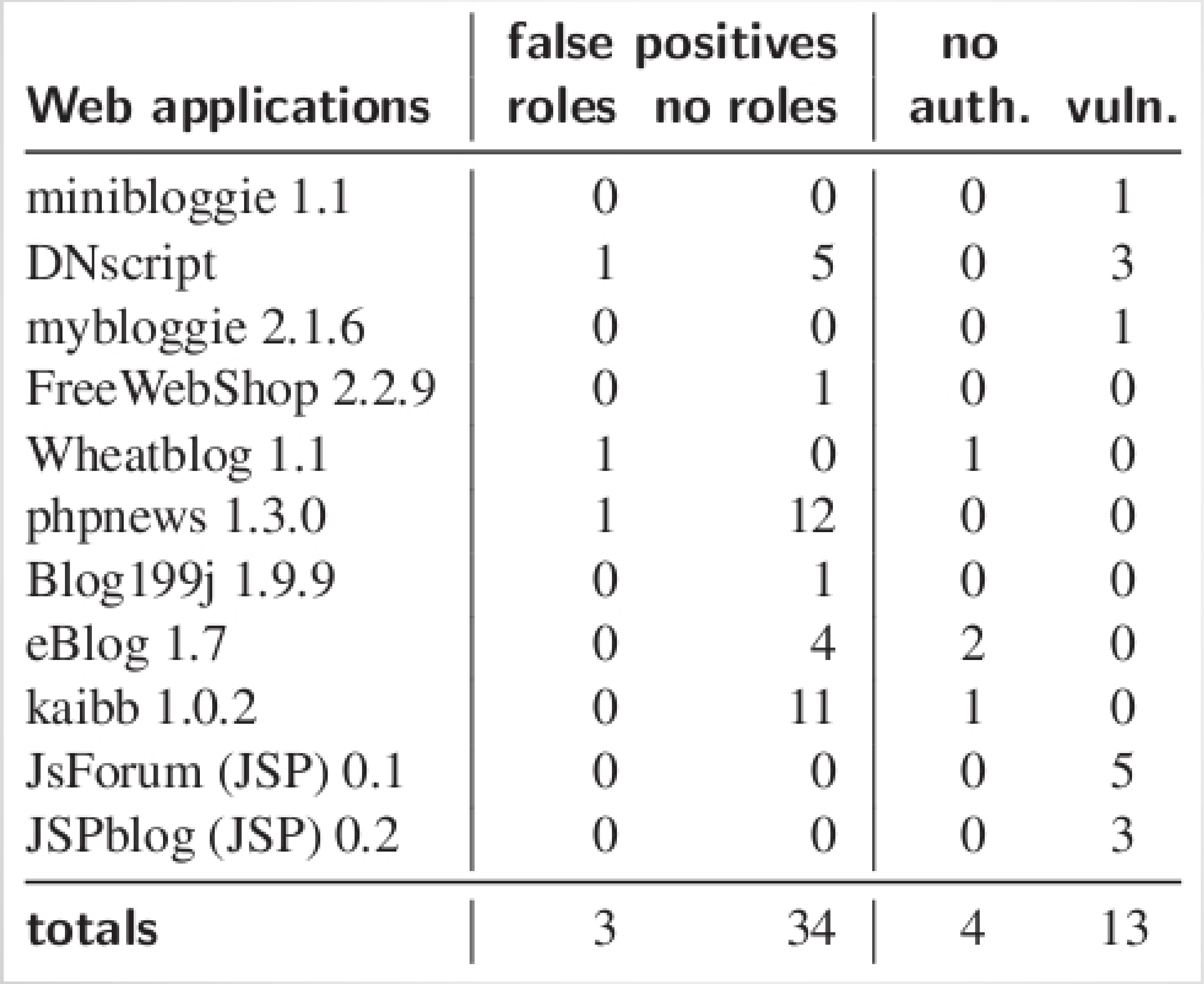
Web applications	LoC	Java LoC	analysis time
minibloggie 1.1	2287	5395	47 sec
DNscript	3150	11186	47 sec
mybloggie 1.0.0	8874	26958	74 min
FreeWebShop 2.2.9	8613	28406	110 min
Wheatblog 1.1	4032	11959	2 min
phpnews 1.3.0	6037	13086	166 min
Blog199j 1.9.9	8627	18749	75 min
eBlog 1.7	13862	24361	410 min
kaibb 1.0.2	4542	21062	197 min
JsForum (JSP) 0.1	4242	4242	52 sec
JSPblog (JSP) 0.2	987	987	16 sec

Experimental Evaluation



Web applications	DB operations (|contexts|)		
	candidates	sensitive	unresolved
minibloggie 1.1	13	3	0
DNscript	99	26	0
mybloggie 1.0.0	195	26	0
FreeWebShop 2.2.9	699	175	0
Wheatblog 1.1	111	30	0
phpnews 1.3.0	80	14	3
Blog199j 1.9.9	195	68	2
eBlog 1.7	677	261	0
kaibb 1.0.2	676	160	0
JsForum (JSP) 0.1	60	32	0
JSPblog (JSP) 0.2	6	3	0

Experimental Evaluation

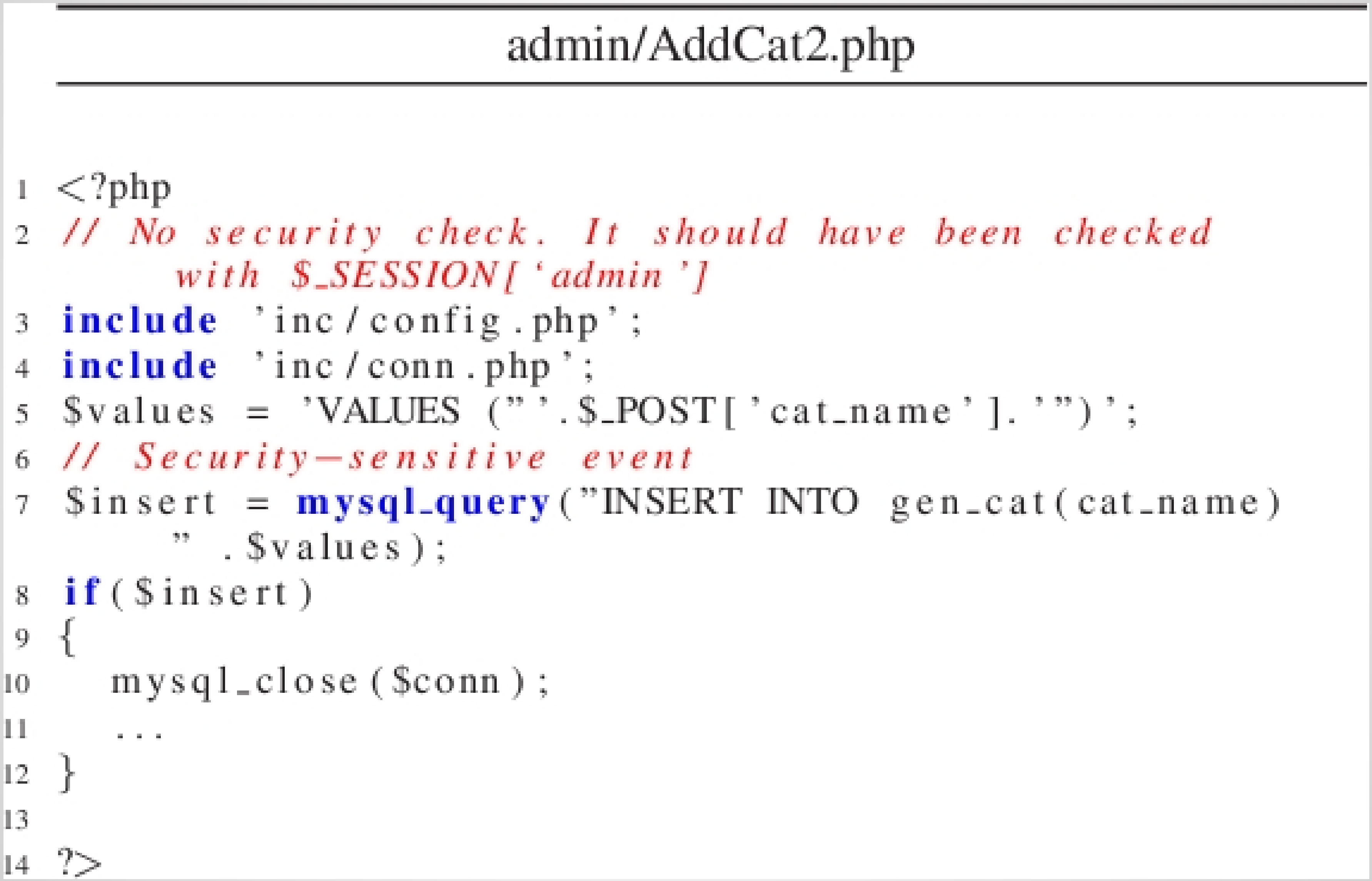


Web applications	false positives roles	false positives no roles	no auth.	vuln.
minibloggie 1.1	0	0	0	1
DNscript	1	5	0	3
mybloggie 2.1.6	0	0	0	1
FreeWebShop 2.2.9	0	1	0	0
Wheatblog 1.1	1	0	1	0
phpnews 1.3.0	1	12	0	0
Blog199j 1.9.9	0	1	0	0
eBlog 1.7	0	4	2	0
kaibb 1.0.2	0	11	1	0
JsForum (JSP) 0.1	0	0	0	5
JSPblog (JSP) 0.2	0	0	0	3
totals	3	34	4	13

Conclusion

- When evaluated on a representative sample of open-source, relatively large PHP and JSP applications, RoleCast discovered 13 previously unreported vulnerabilities with only 3 false positives

Vulnerabilities: DNscript (1)

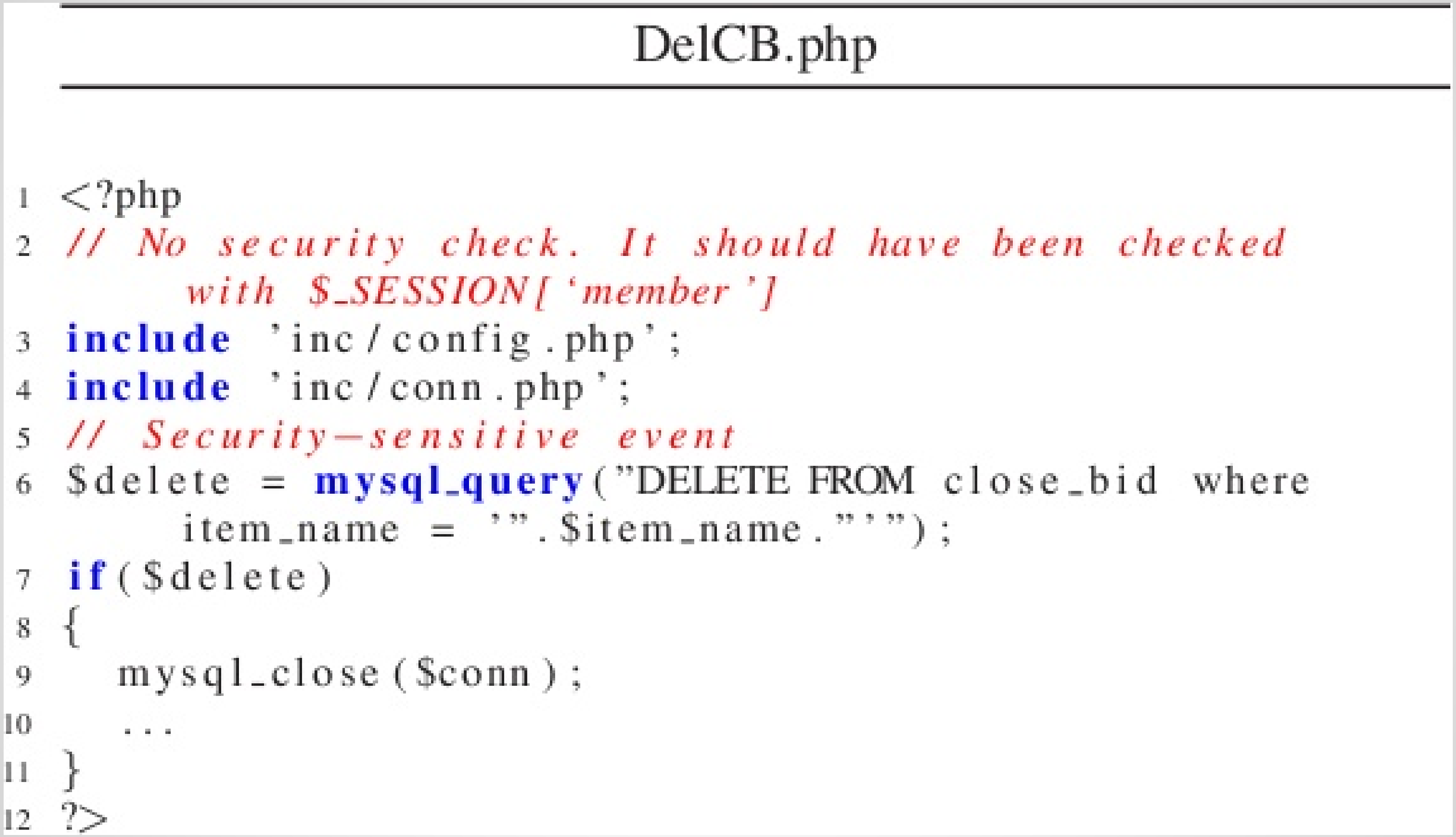


admin/AddCat2.php

```
<?php
// No security check. It should have been checked
       with $_SESSION['admin']
include 'inc/config.php';
include 'inc/conn.php';
$values = 'VALUES ("'.$_POST['cat_name'].'")';
// Security-sensitive event
$insert = mysql_query("INSERT INTO gen_cat(cat_name)
       " .$values);
if($insert)
{
   mysql_close($conn);
   ...
}

?>
```

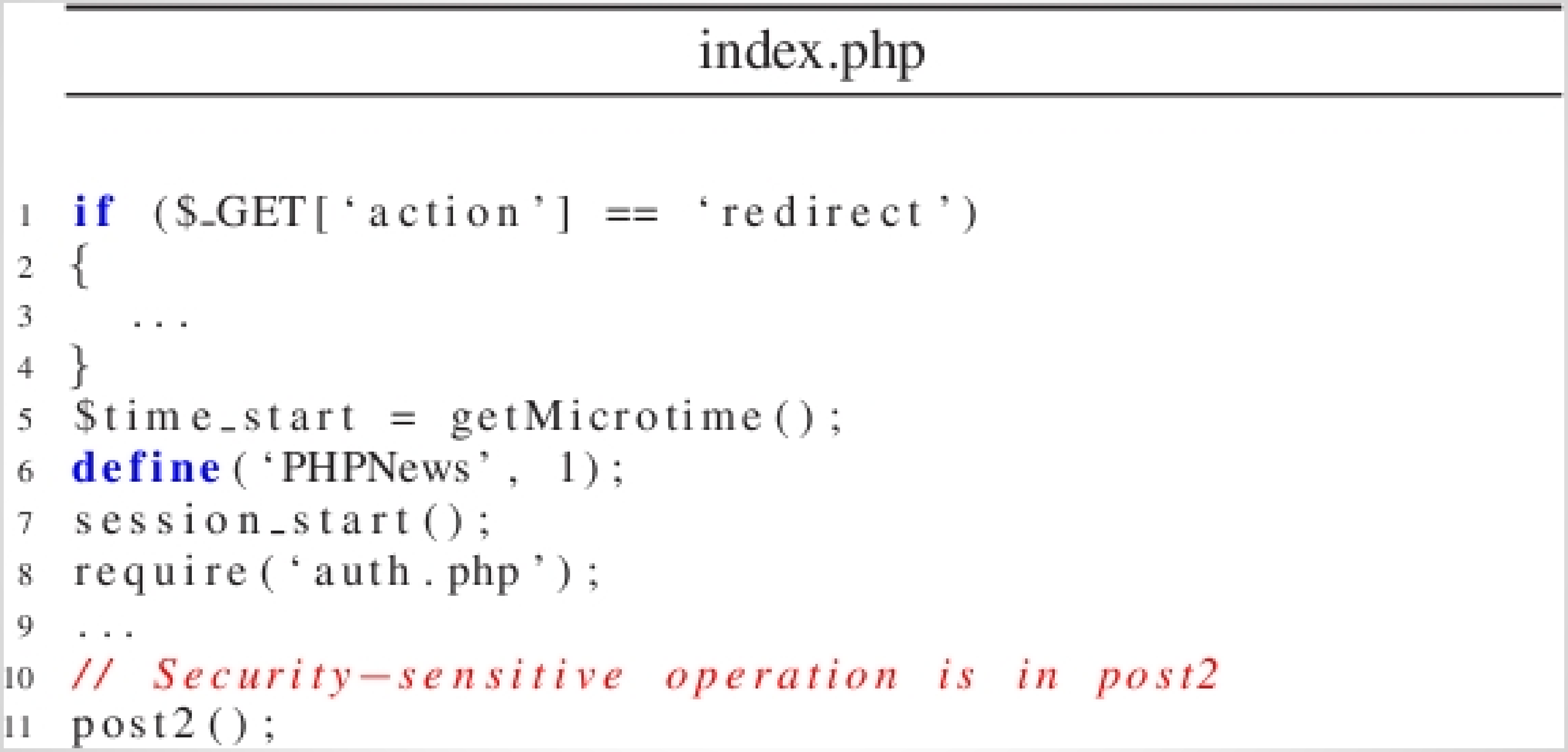
Vulnerabilities: DNscript (2)



DelCB.php

```php
<?php
// No security check. It should have been checked
      with $_SESSION['member']
include 'inc/config.php';
include 'inc/conn.php';
// Security-sensitive event
$delete = mysql_query("DELETE FROM close_bid where
      item_name = '".$item_name."'");
if($delete)
{
   mysql_close($conn);
   ...
}
?>
```

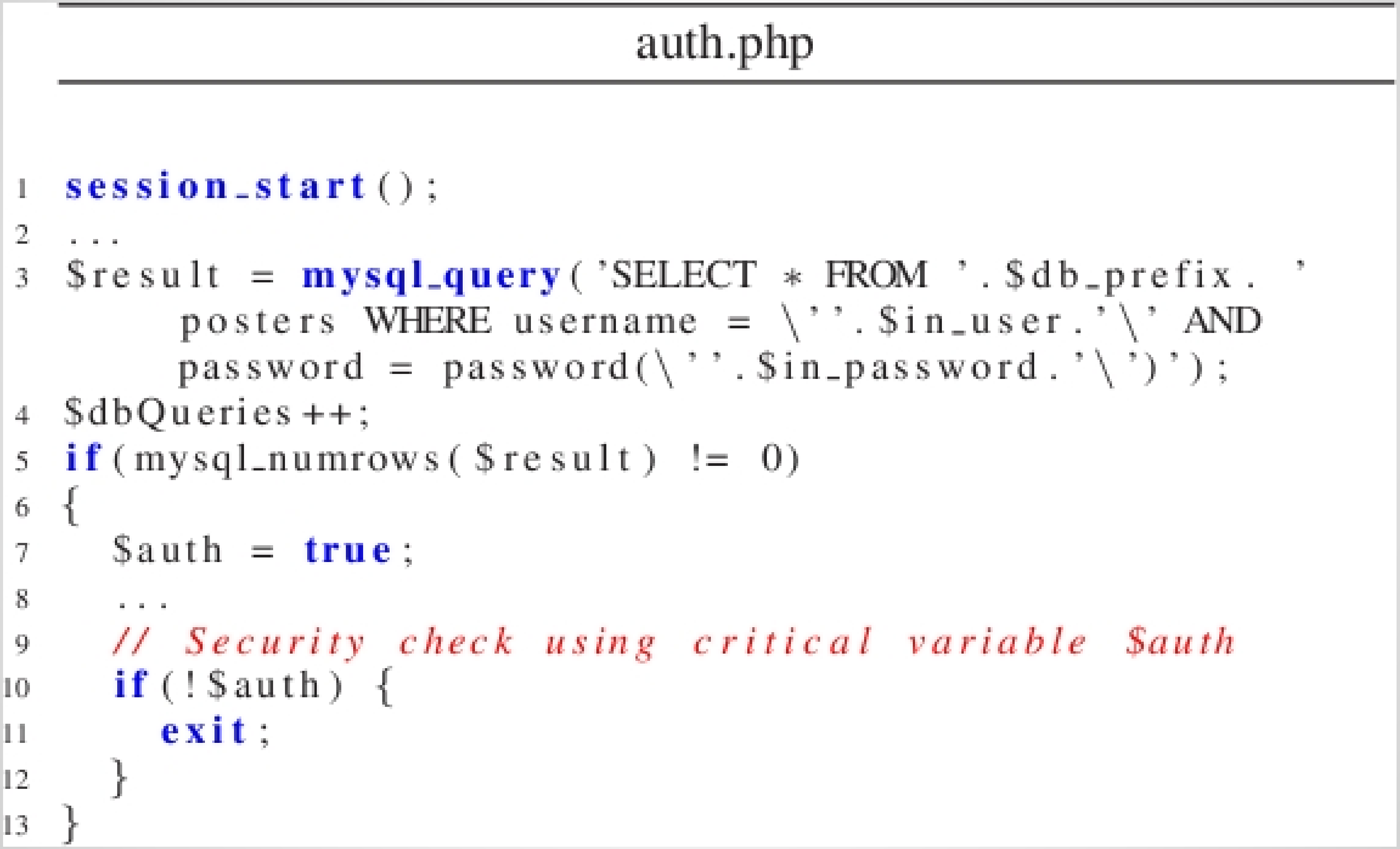
Vulnerabilities: phpnews 1.3.0 (1)



index.php

```
if ($_GET['action'] == 'redirect')
{
   ...
}
$time_start = getMicrotime();
define('PHPNews', 1);
session_start();
require('auth.php');
...
// Security-sensitive operation is in post2
post2();
```

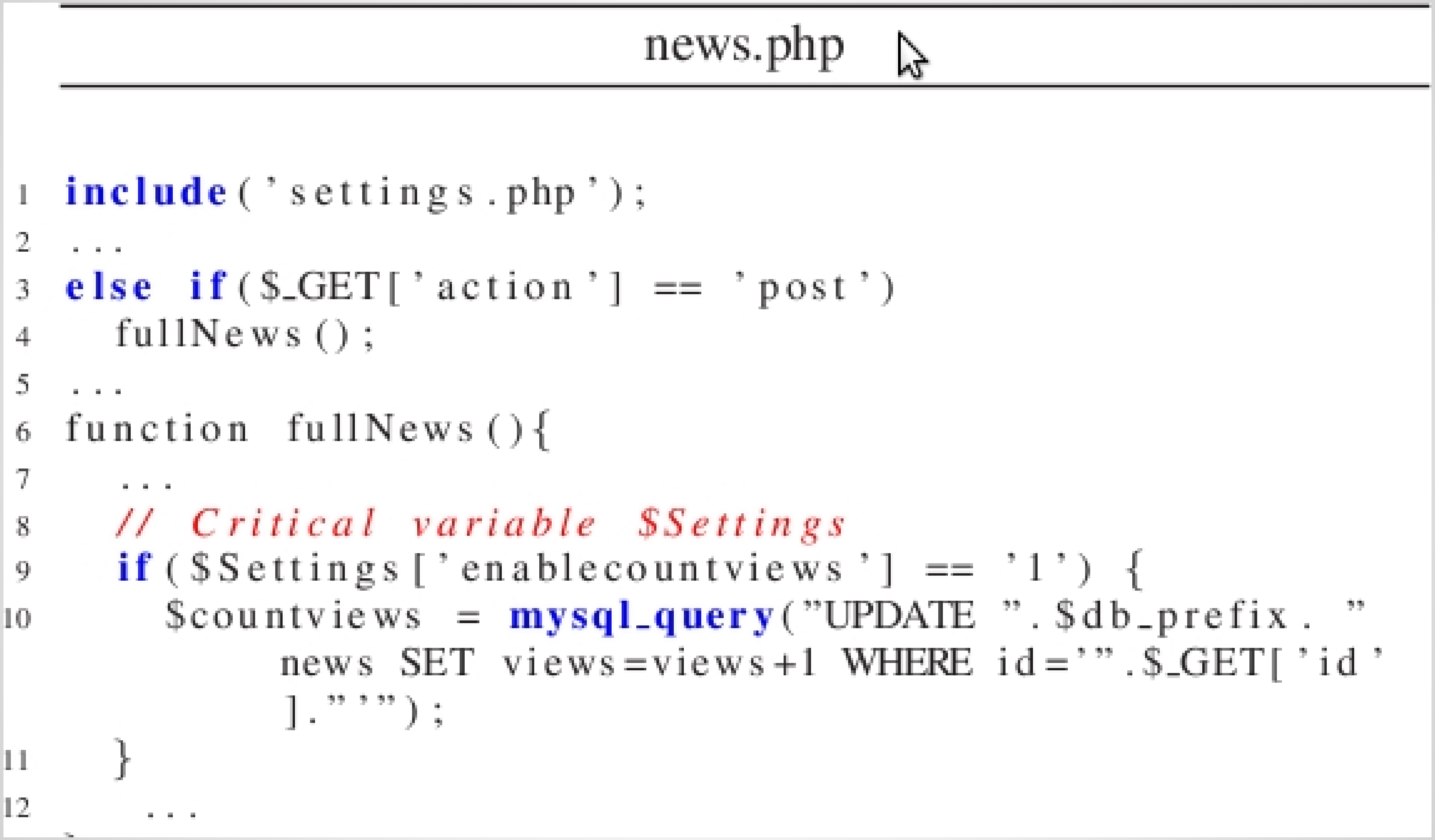
Vulnerabilities: phpnews 1.3.0 (2)



auth.php

```
session_start();
...
$result = mysql_query('SELECT * FROM '.$db_prefix.'
      posters WHERE username = \''.$in_user.'\' AND
      password = password(\''.$in_password.'\')');
$dbQueries++;
if(mysql_numrows($result) != 0)
{
  $auth = true;
  ...
  // Security check using critical variable $auth
  if(!$auth) {
    exit;
  }
}
```

Vulnerabilities: phpnews 1.3.0 (3)



news.php

```
include('settings.php');
...
else if($_GET['action'] == 'post')
  fullNews();
...
function fullNews(){
  ...
  // Critical variable $Settings
  if($Settings['enablecountviews'] == '1') {
    $countviews = mysql_query("UPDATE ".$db_prefix."
         news SET views=views+1 WHERE id='".$_GET['id'
         ]."'");
  }
   ...
```

RoleCast